Python Fun(damentals)

Python Numeric Types

Alexander Rymdeko-Harvey

Obscurity Labs

```
* int
* float
* complex
```

Numeric Types Overview

There are three distinct numeric types:

- intergers - `1`
- floating point numbers - `1.11111`
- complex numbers - `3e+26J`

NOTE: Booleans are a subtype of integers (Eg. `True` or `False`)

For this course, we will be focusing primarily on the basics of `int` and `float` types as your introduced to Python.

Python 3 Types

Its important to understand C/C++ principles first, to understand what makes Python types so easy to work with. Lets take the following example in C:

```c
/* variable definition: */
int a = 1;
char b ='G';
double c = 3.14;
```

Now in Python, notice we dont declare it statically?

```python
# variable definition
a = 1
b ='G'
c = 3.14
```

Python3 Integers

Five standard types you need know:

- Numeric Types - int, float, complex
- Text Sequence Type - str
- Sequence Types - list, tuple, range
- Set Types - set, frozenset
- Mapping Types - dict
- Binary Sequence Types — bytes, bytearray, memoryview

Python Numeric Types

- There are three distinct numeric types: integers, floating-point numbers, and complex numbers
- Booleans are a subtype of integers
- Integers have unlimited precision
- Floating-point numbers are usually implemented using double in C

Integer Operations

Here are a few of the `basic` operations you can perform in Python. This is not an exhaustive list as many operations are out of the scope of this course.

Operation	Result
x + y	sum of x and y
x - y	difference of x and y
x * y	product of x and y

Interger Operations Cont.

Operation	Result
x / y	quotient of x and y
int(x)	x converted to integer
float(x)	x converted to floating point
pow(x, y)	x to the power y
x ** y	x to the power y

Using Intergers

Using the python interpreter we can perform live math operations:

```
Python 3.7.5 (default, Apr 19 2020, 20:18:17)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 + 10 <-- basic math
20

>>> type(10 + 10) <-- use type() to check the type
<class 'int'>

>>> int(10 + '10') <-- will fail as a string is not a int
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Using Intergers Cont.

Using the python interpreter we can also use the very first function we will learn `print()`. This prints the value of the `sum`:

```
Python 3.7.5 (default, Apr 19 2020, 20:18:17)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(10 + 10) <-- basic math notice as its the same
20
```

Using Floats

Take not the differences between `float` and `int` when we call the `int()` function:

```
Python 3.7.5 (default, Apr 19 2020, 20:18:17)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 10.5 + 10.5
21.0

>>> type(10.5 + 10.5)
<class 'float'>

>>> int(10.5 + 10.5)
21

>>> float(10.5 + 10.5)
21.0
```

Using Floats and Ints

Take not the order of operations and combination of an `int` and `float`

```
Python 3.7.5 (default, Apr 19 2020, 20:18:17)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 + (10.5 + 10.5)
31.0

>>> type(10 + (10.5 + 10.5))
<class 'float'>

>>> exit()
```

Lab 1 - Familiarization

Tasking

Using the new `python` command perform the following get familiar with `numeric` types:

1. Perform a basic addition operation ex. `10 + 10`
2. Perform a basic `float` operation ex. `10.1 + 10.21`
3. Perform a basic order of operations ex. `10 + (10.5 + 10.5)`

Python Numeric Built-in Methods

Many of the `Types` within Python have methods and in your IDE such as VSC will allow you to explore these.

Here is a example of `int.bit_length()`:

```
>>> n = -37
>>> bin(n)
'-0b100101'
>>> n.bit_length()
6
```

NOTE: bin() is the binary representation*

Here is a example of `int.to_bytes()`:

```
>>> (1024).to_bytes(2, byteorder='big')
b'\x04\x00'
```

Lab_1.py

TASKING

Perform the following on the variable `dataNum`:

1. Set Value to `1.2299`
2. Set `dataNumPower` to power of 2 for `dataNum`
3. Set `dataNum2` to the int `10`
4. Set `dataNum2Bytes` to the bytes of `dataNum2`, setting the length to `1` and the byte order to `big`